Abstract: - One of the most active research areas in tissue engineering deals with the modification of scaffold materials and surfaces in order to enhance tissue growth rates. This paper presents the parallelization of a three-dimensional computational model for wound healing based on cellular automata. The model is used to study the effectiveness of scaffold surface modification for increasing tissue growth rates through enhanced cell migration speeds. Issues related to the parallelization of the model on a distributed-memory multi-computer such as correctness and determinism are discussed. Both the sequential and parallel simulation results are presented and compared.

Key-Words: - Cellular automata, Correctness, Determinism, Distributed memory, Parallelization, Tissue engineering, Wound healing.

1 Introduction

In health care, one of the most costly and devastating problems is the loss or failure of an organ or tissue. Currently, physicians apply therapies based on organ transplantation, surgical reconstruction, or the use of mechanical devices. While helpful in many cases, such therapies are considered to be imperfect solutions [7]. One alternative approach is to replace the failing tissue or organ with a biological substitute grown from the patient’s own cells. Bio-artificial constructs aiming to reproduce the architecture of the tissues they replace, must be built on three-dimensional scaffolds made from suitable biomaterials capable of playing multiple roles in tissue-engineered constructs. Besides serving as the structural component, biomaterials provide the proper architecture of the construct, a three-dimensional matrix for guided cell migration and proliferation, and growth factors to induce and maintain differentiated cell function.

Three-dimensional scaffolds may also be used to promote wound healing, a serious problem with patients suffering from many debilitating diseases [8]. According to this approach, a biocompatible matrix is used to fill the defect (wound). This matrix contains bioactive agents and growth factors that induce neighboring cells to migrate into the scaffold, proliferate, and produce their own extracellular matrix. This process continues until the wound heals.

Scaffold properties such as surface adhesiveness and the presence of growth factors, cell activities like adhesion or migration, and external stimuli that modulate cellular functions are among the many factors that affect the growth rate of tissues. Hence, the development of bio-artificial tissue substitutes involves extensive and time-consuming research and experimentation. The availability of computational models with predictive abilities will greatly speed up progress in this area. Such tools will assist researchers in predicting the dynamic response of cell populations to external stimuli, and in assessing the effect of various system design parameters on the overall tissue growth rates.

In an earlier communication, we presented the development of a new sequential model for the growth of three-dimensional tissues and wound
healing [2]. Our interdisciplinary research group has been working for the last several years on building cellular automata based simulation tools for tissue engineering [3, 16].

2 Development of the Model

We consider cellular automata [1, 15] consisting of three-dimensional grids with \( N \) cubic computational sites. Each site is a finite automaton that can exist at one of a finite number of states at each time interval. That is, a site may be either:

(a) empty and available for a cell to move in, or
(b) occupied by a cell, which is at a given point in its mitotic cycle and moves in a certain direction.

The state of each automaton evolves at discrete time steps through interactions with neighboring automata. For any automaton \( i \) \((1 \leq i \leq N)\) where \( N = N_x \times N_y \times N_z \), there exists a local transition function specifying the state \( x_i(t+1) \) of the automaton \( i \) at time level \( t+1 \) as a function of the states of its neighbors at time level \( t \):

\[
x_i(t+1) = f_i(x_i(t), x_{i+v_i}(t), \ldots, x_{i+v_{i_z}}(t)),
\]

where \( i = 1,2,\ldots,N \) and \( x_{i+v_i}, x_{i+v_{i_1}}, \ldots, x_{i+v_{i_z}} \) are the states of the neighbors of automaton \( i \). The local transition functions must be carefully chosen to model persistent random walk of cells, cell-cell interactions, and cell divisions. The simultaneous application of the local transition functions to all the automata in a cellular space transforms a configuration \( X(t) = (x_1(t), x_2(t), \ldots, x_N(t)) \) of the cellular automaton to another one \( X(t+1) \) according to Equation (1). Thus, a global transition function \( F \) acting on the entire cellular space can be defined as:

\[
X(t+1) = F(X(t)) \text{ for } t = 0,1,2,\ldots
\]

Starting from an initial configuration \( X(0) \), the cellular automaton follows a trajectory of configurations \( X(1), X(2), \ldots, X(t), \ldots \) defined by the global transition function \( F \). At each time level, the states of all the cells of an automaton are updated in parallel [3]. In the 3-D model, each computational site interacts with six immediate

neighbors located to its north, east, south, west, above, and below it. This is an extension of the von Neumann neighborhood to three spatial dimensions.

Proliferating cells execute persistent random walks in space. This process consists of the following stages:

1. Each cell in the population moves in one direction for a certain period of time (persistence). At the end of this interval, the cell stops and turns to continue its migration in another direction. The persistence is a random variable whose density function can be determined experimentally.
2. When two cells collide, they stop for a short period of time before resuming their migration to move away from each other.
3. At the end of its cycle, a cell stops to divide into two daughter cells. The cell cycle or division time is another random variable whose density function can be experimentally measured.
4. This process is repeated until the cell population has completely filled the scaffold or until the cells cannot migrate and divide any further.

To simulate these dynamics, the state \( x_i(t) \) of each cellular automaton takes values from a set of integer numbers that code all the required information about cell migration speed, the direction of movement, and the time remaining until the next direction change and the next cell division. This information can be obtained from independent experiments using the locomotion assays developed in [9].

2 Parallelization of the Model

The parallelization of the model is motivated by the fact that the simulation of realizable tissue objects requires very large amounts of memory. To illustrate the extent to which our application is memory intensive, we estimate that simulating a 1-cm\(^3\) tissue object requires 26 Giga Bytes (GB) of memory sequentially and 32 GB in parallel ((\(32/P\)) GB are needed per node on a parallel machine with \( P \) processors).

Our application falls in the category of loosely synchronous ones [13]. Such applications exhibit certain characteristics where the amount of computation could vary from one partition and time step to another because it depends on the amount of useful data. Since the workload varies both temporally and spatially, much effort must be spent
To evenly distribute it among the nodes. Otherwise, we risk an increase in overhead, which in turn can decrease performance. To mitigate these problems, we used a static load-balancing strategy that preserved the initial cell distribution as outlined below.

2.1 Domain Decomposition
The “wound” was modeled as a cylinder in the center of the cubic computational grid and assumed that it was completely filled with scaffold material (see Fig.1). The decomposition was achieved along the z dimension only. This is called the slab decomposition. The mapping of the cellular array onto P processors is as follows. If P divides \( N_z \), then all the sub-domains have equal size consisting of \( N_x \times N_y \times \frac{N_z}{P} \) sites. If, however, \( N_z = qP + r \), then the array is divided into \( r \) sub-domains of size \( N_x \times N_y \times \left( \frac{N_z}{P} + 1 \right) \) and \( P - r \) sub-domains of size \( N_x \times N_y \times \left( \frac{N_z}{P} \right) \). The area of the boundary between any two sub-domains is equal to \( N_x \times N_y \) sites. Except for the sub-domains at both ends of the decomposition, all remaining sub-domains have two neighbors. The cellular space is mapped onto a one-dimensional processor grid. This logical topology provides a convenient naming mechanism for the processes of a group and reflects the logical communication pattern of the processes in a parallel machine.

Fig.1: Initial cell seeding showing the wound-healing mode and a slab decomposition on \( P = 4 \) processors along \( N_z \).

2.2 Correctness
Our objective is to solve the same problem with the parallel code as with the serial code. Two algorithms \( A_1 \) and \( A_2 \) may give somewhat different cell population distributions after a certain number of steps, but both are correct if the distributions were achieved by legal processing of the rules of our automaton. The problem is defined by a set of allowed cell movements and divisions within a defined neighborhood and according to state information. A possible input is an initial cell population distribution where each occupied site is assigned a state as mentioned previously. The corresponding set of possible outputs consists of all cell population distributions obtained, after some time-steps, according to the defined rules of cell proliferation and migration. One of the rules is the collision rule specifying that no two cells are allowed to occupy the same site. Sequentially, this does not cause any problems. A single node owns the entire cellular space and knows its state. Before allowing cell movement and division, it can test whether such processes are permitted. If so, it marks the prospective site. If not, it looks for another unoccupied site within the cell’s neighborhood. At the end of each step, it updates the entire cellular array.

In parallel, the problem domain is decomposed into sub-domains owned by different nodes. The processing of occupied sites at ownership boundaries becomes difficult. Cells crossing borders, either by movement or division, is the type of ownership change that can cause correctness problems. Here, the boundary areas are shared by up to two neighboring nodes (see Fig.2). To preserve the semantics of the serial algorithm, in the parallel algorithm cells must be allowed to cross borders either due to motion or division and no two cells should be allowed to occupy the same site. Therefore, an appropriate collision resolution technique must be employed to update the state of cells in each boundary area.
The idea here is to split the parallel algorithm into two different phases, calculation and execution phases. Between these two phases, we have the opportunity to ensure the correctness of the algorithm. If we perform these two phases in one operation without communicating with the neighboring nodes, we lose correctness. For instance, by applying this splitting technique, each parallel simulation step of cell motion involves:

(a) exchanging shared boundaries with neighboring processors,
(b) calculating cell movements,
(c) exchanging computed movements in shared boundaries with neighbors,
(d) resolving collisions due to motion of cells,
(e) exchanging the results of collision resolution due to motion of cells in shared boundaries (Step (d)) with neighboring nodes, and
(f) executing the motion of cells by taking into account the local calculations and the collected ones from neighbors.

Similar steps can also be obtained for cell division. The attainment of this goal was enhanced by the following two observations:

1. The resolution of collisions between cells at the boundary was done in an unbiased manner so that the update of the states of these cells may be retried by following the same cellular automata rules.
2. Cells are able to cross over processor boundaries as well as make non-boundary moves and divisions.

### 2.3 Determinism

Determinism is related to obtaining reproducible results for the same set of input data. Deterministic results are needed for the following purposes:

- **Verification:** To get the same results whether using a parallel system with two processors or one with 16 processors is a strong argument for the correctness of these results.

**Comparison:** To be able to compare the performance of multiple systems, ensuring that similar results are obtained on these systems is paramount.

The key to determinism is a precisely given advice for how to make decisions on cell motion or division. Such decisions are based on the persistence and division time random variables. The implementation of these random variables calls for the use of transition state probabilities, growth probabilities, and a random number generator (RNG). We opted to implement a RNG based on *Lehmer’s method* [10] due to its good and well-understood statistical properties, namely, randomness and periodicity [3, 11]. In addition, such generator gives the user control over the choice of the seed value, thus maintaining its deterministic behavior. Serially, the desired deterministic behavior is obtained because the order in which data are processed is only defined by the algorithm and not influenced by any external factors.

In parallel, determinism is difficult to maintain due to at least two factors: The first factor deals with how the decomposition of the problem domain is done. This becomes very complicated if dynamic load-balancing strategies are used. The second factor deals with hardware. For MIMD (Multiple Instruction Multiple Data) machines, each node has its own clock, and there are always slight speed differences. For example, if two nodes want to communicate with a third node at the same time, we cannot predict which node will be serviced first. Given these issues, it is hard to produce a stable parallel code which will always produce the same results. The basis of our solution is twofold. First, we use a static load-balancing strategy that is dependent on the initial cell distribution as discussed previously. Second, we parallelized the RNG in such a way that keeps our computation deterministic even if the order of the generation of the parallel subsequences by the different nodes is not fixed. We used the *leaping* strategy outlined in [6]. Given a seed for a serial simulation run, deterministic and unique parallel seeds were then obtained for the parallel simulation runs.

### 4 Results and Discussion

To simulate the wound healing process, we generated a cylindrical “wound” with a diameter equal to 64 sites and a height equal to 128 sites. We then allowed cells from the surrounding fully-
developed tissue to migrate into the wound area and proliferate.

The model was implemented both sequentially and in parallel on a distributed-memory machine, the IBM SP2, available at the University of Houston’s Texas Center for Computational and Information Sciences (UH-TCCIS). The Single Program Multiple Data (SPMD) programming model and the Message Passing Interface (MPI) library were used in the parallel implementation [5, 12]. Simulations were carried out on a 128x128x128 computational grid with cell speeds varying across a wide range of values: from non-motile cells to 50 µm/hr. For the parallel simulations, the number of processors \( P \) varied from 2 to 32 nodes. We present simulation results for parallel runs on \( P = 4 \) processors.

Both serial and parallel simulation results (see Fig.3-Fig.6) show enhancements of tissue growth rates and significant reduction of wound healing times with increasing cell migration speeds. These results are important, for they point out that cell population dynamics and initial conditions can have a profound impact on tissue growth rates.

Both serial and parallel simulation results (see Fig.3- Fig.6) show enhancements of tissue growth rates and significant reduction of wound healing times with increasing cell migration speeds. These results are important, for they point out that cell population dynamics and initial conditions can have a profound impact on tissue growth rates.

**5 Comparison of Simulation Results**

The following table shows the comparison between the times to grow new tissue and reach confluence obtained by the serial and parallel implementations for different cell migration speeds. The relative time differences with respect to the times reached by the sequential algorithm for each corresponding pair of values are also included. We notice that the differences are small meaning that the parallel results are very close to the serial ones, thus, validating the splitting technique that we used. This is confirmed further by Fig.7, which displays simultaneously both the serial and parallel results of new tissue growth rate for cell migration speeds of 5 and 50 µm/hr, respectively. We see that the macroscopic behavior of the parallel results is identical to the serial ones.
Table 1: Comparison of the time in days to grow new tissue between the serial algorithm and its parallel implementation for different cell speeds. The relative difference between each pair of times is also indicated.

<table>
<thead>
<tr>
<th>Cell Speed in µm/hr</th>
<th>Time in Days (Serial)</th>
<th>Time in Days (Parallel)</th>
<th>Relative Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>26.250</td>
<td>26.250</td>
<td>0%</td>
</tr>
<tr>
<td>1.667</td>
<td>15.500</td>
<td>15.500</td>
<td>0%</td>
</tr>
<tr>
<td>3.333</td>
<td>10.500</td>
<td>10.500</td>
<td>0%</td>
</tr>
<tr>
<td>5</td>
<td>8.167</td>
<td>8.250</td>
<td>1.02%</td>
</tr>
<tr>
<td>10</td>
<td>5.875</td>
<td>5.917</td>
<td>0.71%</td>
</tr>
<tr>
<td>50</td>
<td>2.842</td>
<td>2.849</td>
<td>0.25%</td>
</tr>
</tbody>
</table>

Fig.7: Comparing the new tissue growth rate of the serial and parallel implementations for two different cell migration speeds.

6 Conclusion

We reported in this paper the parallelization of a three-dimensional computational model for wound healing. Based on the concept of cellular automata, the model allowed us to explore the impact of cell migration speeds on new tissue growth rates. Our parallel implementation ensured that correct and deterministic simulation results were obtained. This will enable us to simulate the growth of realizable tissue objects on readily available parallel machines such as Beowulf clusters and to proceed with planned extensions of present research.

References: